Beyond Pruning and Dropout: Evolving Robust Networks via Persistent Stochastic Ablation

Tim Cotten
Scrypted Inc., George Mason University
tcotten@scrypted.ai, tcotten2@gmu.edu

***Abstract*—This paper introduces and empirically evaluates "Persistent Stochastic Ablation (PSA)," a novel training paradigm that simulates evolutionary pressure via a ratcheting meta-learning process. At each cycle, PSA corrupts a pristine copy of the network's best-performing state (its "Last Known Good" state) through random neuron ablation. This damaged model is then trained against the last best performance benchmark on the validation dataset. A successful outcome promotes the newly trained model to become the next LKG state, effectively "ratcheting" the performance baseline upwards. This "frustration" mechanism forces repeated recovery attempts from a disadvantaged state, driving the model to escape difficult local optima or find more robust regularization solutions. Using a custom meta-learning harness, we conduct an systematic study on feed-forward Multi-Layer Perceptron (MLP) architectures "SimpleMLP" across a landscape of varying width, depth, and parameter counts on the MNIST dataset. Our findings reveal that the efficacy of PSA is not universal but is highly contextual. We trained 98 model configurations with six modes in a 3 vs. 3 comparison of baseline regularization techniques versus PSA: {'None' (Control), 'Weight Decay', 'Dropout'} and {'Full', 'Hidden', 'Output'}. For each combination we performed 10 randomly seeded trainings for a total of 5,880 total trial results. We identified and characterized four distinct behavioral training regimes: Beneficial Regularization, Optimally Sized, Chaotic Optimization, and Architectural Failure. These regimes are dictated by the interplay between model capacity and architectural stability; we document the regimes where PSA methods are beneficial, harmful, or indistinguishable compared to the existing control or regularization techniques, such as in over-parameterized models where 'Dropout' is highly effective. Further, we find a non-regularization use of PSA as a stochastic "kick" for low-capacity or exotic models to escape local minima that weight decay and dropout cannot rescue. By analyzing the temporal dynamics of model improvement, we demonstrate that PSA can act as a sustained, exploratory search mechanism, capable of achieving late-stage breakthroughs in otherwise difficult or intractable optimization landscapes. Crucially, this initial study is limited by the fundamental restrictions of the `SimpleMLP` architecture due to the boundaries of the Vanishing Gradient Problem, beyond which deeper topologies become untrainable, thereby establishing the necessary groundwork for future research with more robust architectures like a ResMLP with skip connections.**

***Index Terms*—Neural Network Regularization, Ablation Studies, Evolutionary Algorithms, Deep Learning, Vanishing Gradient Problem, Local Minima, Model Capacity, Model Stability, Multi-Layer Perceptron (MLP), Stochastic Optimization, Meta-Learning, Network Robustness, Fault Tolerance**

Conflict of Interest Statement: Tim Cotten is an employee of Scrypted Inc. and has a financial interest in the company. This work is presented as foundational research, and its findings may inform future technology development at Scrypted Inc..

I. Introduction

Deep neural networks, despite their demonstrable power, remain susceptible to two fundamental issues: becoming trapped in poor local minima during optimization, and a tendency to learn brittle feature representations due to overfitting in over-parameterized architectures. Conventional ablation methodologies to combat these challenges fall into two dominant paradigms: **post-hoc pruning** [1], [2] for computational efficiency, and **transient regularization**, like Dropout [3], to prevent feature co-adaptation during training.

This study investigates a third paradigm of ablative techniques in deep learning we term **Persistent Stochastic Ablation (PSA)**. Unlike its predecessors, PSA is not motivated by efficient reduction of network size or preventing immediate co-adaptation of features. Nor does it rely on post-hoc analysis and pruning. Instead, it leverages *blind, iterative harm* as a continuous, online training driver.

We hypothesize that by repeatedly forcing a network to recover from non-strategic damage to its best-performing state, we can simulate a virtual evolutionary pressure, and that this pressure can manifest in at least two ways: first, by acting as a powerful stochastic search mechanism, capable of "kicking" ill-conditioned models out of poor local minima, and second, by encouraging more robust and fault-tolerant feature representations. This concept is inspired by Neural Darwinism [4], wherein stochastic variation and selection drive adaptive neural development.

To test this hypothesis, we developed the "Frustration Engine," a meta-learning framework that implements the PSA training loop. The process is a simple, powerful ratchet: 1) A pristine copy of the network's best state (the "Last Known Good" or LKG) is corrupted via random neuron ablation. 2) This damaged model is trained against the LKG's performance benchmark. 3) If successful, the new state becomes the next LKG.

This "meta-loop" design continuously forces the model to recover from a disadvantaged state, fundamentally differing from Dropout, which applies transient noise that is discarded after each weight update within a single training epoch. In contrast, the "damage" and "recovery" from PSA is persistent

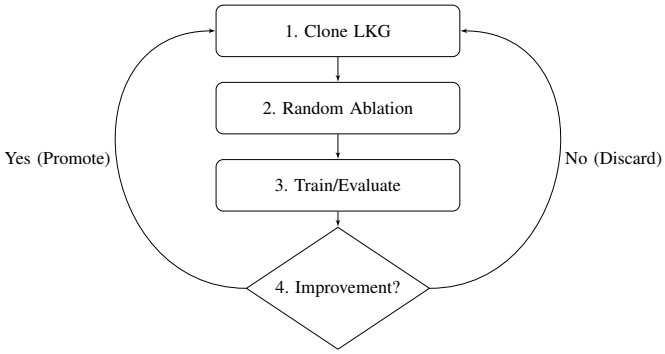

Fig. 1. The conceptual flow of the Persistent Stochastic Ablation (PSA) meta-loop. On failure, the process discards the candidate and starts a new cycle from the unchanged LKG state. On success, the challenger is promoted to become the new LKG.

between successful training cycles, forcing a structural, rather than merely statistical, adaptation.

Utilizing an outer loop that guides an inner learning process is a cornerstone of meta-learning, established by works like Model-Agnostic Meta-Learning (MAML) [6] for few-shot adaptation. Our proposed strategy is thus inspired by MAML-like meta-learning paradigms, not for cross-task adaptation, but for guiding the optimization of a single model on a single task despite the "frustration" of iterative corruption, using a ratcheting mechanism to capture learning opportunities.

This initial study documents our exhaustive experiments on a simple Multi-Layer Perceptron (MLP) architecture using ReLU activation neurons, mapping its response to PSA across a wide range of architectures defined by varying capacity and stability. This study seeks to answer three fundamental questions:

1. Under what conditions, particularly in cases of architectural instability or optimization failure (e.g. the Vanishing Gradient Problem [5]), can PSA provide a unique benefit by enabling escape from otherwise intractable local minima?

2. In what regimes does PSA act as a competitive regularizer, and when does it become an irredeemably detrimental source of damage compared to established techniques like Weight Decay or Dropout, or even the control?

3. Can we demonstrably disentangle the effects of online neural ablation via PSA depending on the affected layers, specifically differentiating and measuring the effects of stochastically operating on hidden layers, the output layer, or all of them combined?

II. Methodology

Our methodology is designed to rigorously evaluate Persistent Stochastic Ablation (PSA) across a wide experimental landscape. All trials were conducted within our "Frustration Engine" meta-learning framework to ensure controlled and fair comparisons between training paradigms. The experiment is structured along two orthogonal axes. The first axis compares three PSA modes (`Full`, `Hidden`, and `Output`) against a baseline group of three standard training methods: a no-intervention control (`None`), `Weight Decay`, and `Dropout`. The second, architectural axis tests these six modes across a curated landscape of 98 distinct Multi-Layer Perceptron (MLP) topologies on the MNIST dataset. Each of the resulting 588 combinations was trained 10 times with independent random seeds, yielding 5,880 total trials. This large-scale design provides the statistical power necessary to move beyond anecdotal results, allowing us to isolate the influence of model capacity and architectural stability, and to characterize the distinct behavioral regimes that emerge from the interplay between each strategy and the network it attempted to train.

A. Experimental Landscape

To disentangle the effects of network topology from raw parametric capacity, we designed our experimental landscape around the principle of **parameter matching** and **mirrored configurations**. Using our experimental `SimpleMLP` testbed, we curated a collection of 98 homogeneous architectures (L*W) that systematically explore the 2D space of network depth (L) and width (W), as visualized in Figure 3.

First, we created a distribution of architectural configurations of `1*W` networks, where $2 \leq$ `W` ≤ 2048. We then extended this to deeper "shallow-and-wide" configurations (e.g., `2*W`, `4*W`) by solving for a width $\boldsymbol{W}$ that approximately matched the parameter count of a `1*W` counterpart (ex.{`1*2048`, `2*939`, `4*632`} having {`1,628,170`, `1,629,175`, `1,702,618`} parameters).

To do this, we first used the standard formula for determining the parameters of an MLP:

$$P_{total} = H_1(N_{in}+1)+\sum_{i=1}^{L-1} H_{i+1}(H_i+1)+N_{out}(H_L+1) \quad (1)$$

where $N_{in} = 784$ and $N_{out} = 10$. Then, we solved for a width approximation formula based on the target parameter count $\boldsymbol{P}$ and fixed number of layers $\boldsymbol{L}$:

$$W_{\text{target}} \approx \frac{-(794+L_{\text{target}})+\sqrt{(794+L_{\text{target}})^2+4(L_{\text{target}}-1)(P_{\text{base}}-10)}}{2(L_{\text{target}}-1)} \quad (2)$$

Second, we created "square" network architectures of shape `L*W` where `L=W` in order to explore deeper topologies, still solving for parameter matching using the previous formula (ex. `115*115` being 1,612,195 parameters to match `1*2048`).

Third, we mirrored the "shallow-and-wide" network configurations into their inverted "deep-and-narrow" opposites, such as adding `16*4` to complement `4*16`. We deemed these **asymmetric**, as we couldn't feasibly design parameter matching equivalents along the mirrored axis, opting instead to invert the `L*W` relationship to `W*L` (ex. we added `256*1` to complement `1*256`). We did this with the full understanding that such pathological networks were not viably trainable, but felt it was valuable to experiment on them nonetheless.

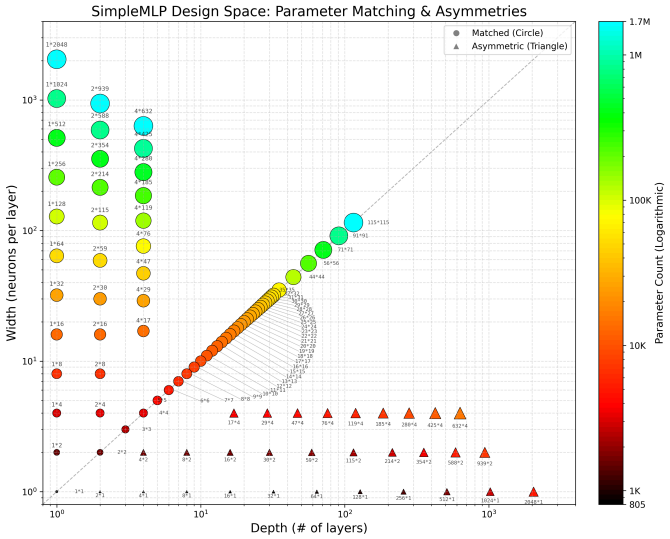

Fig. 2. The architectural design space, illustrating our principles of parameter matching and asymmetry. Architectures in the shallow-and-wide (top-left) and square (diagonal) regions are parameter-matched (circles), with size indicating total parameters. Asymmetric deep-and-narrow configurations (triangles) are included to explore the boundaries of trainability.

Finally, in order to identify the behaviors of the baselines and ablative techniques as deeper networks failed due to the Vanishing Gradient Problem (VGP), we created additional parametrically asymmetric square designs to fill the entire space between `1*1` and `32*32`.

The full test suite is documented in the Appendix: Table IV

B. The SimpleMLP Testbed

The experimental testbed for this study is a feed-forward Multi-Layer Perceptron (MLP), termed `SimpleMLP`, designed to be dynamically configured from a single command-line argument (`--arch`).

`SimpleMLP` is constructed as a sequence of hidden blocks, where each block consists of a **Linear** layer, followed by a **Rectified Linear Unit (ReLU)** activation, and a **Dropout** layer. Blocks are defined in the form `L*W` where `L` represents the number of layers and `W` the width of each layer in neuron count. The network's final layer is a linear output head that maps the final hidden state to 10 output logits for MNIST classification. The forward pass for an input $\boldsymbol{x}$ and $\boldsymbol{L}$ hidden layers is defined as:

$$\begin{aligned} h^{(0)} &= \text{Flatten}(x) \\ h^{(i)} &= \text{Dropout}\big(\text{ReLU}(W_i h^{(i-1)} + b_i)\big), \quad \text{for } i = 1, \ldots, L, \\ y_{\text{logits}} &= W_{\text{out}} h^{(L)} + b_{\text{out}}. \end{aligned} \tag{3}$$

where $h^{(i)}$ is the output of the i-th hidden block, and W and b represent the weight matrices and bias vectors for their respective layers.

To ensure a fair comparison, `nn.Dropout` modules are included in the model graph for all configurations. These layers are only active during training when `--ablation-mode` is explicitly set to `dropout`; in all other modes, they are set to evaluation mode and function as identity operators. We do not include them when calculating parameters, and this design guarantees that the number of trainable parameters remain identical across all experimental conditions: isolating the behavioral effects of the chosen regularization or ablation strategy within any given architectural design.

Thus, a given MNIST training architecture can be summarized as:

$$\begin{aligned} &\text{Input (784-dimensional)} \\ &\qquad \rightarrow [\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout}] \times N \\ &\qquad \rightarrow \text{Linear (10 neurons)} \rightarrow \text{Output} \end{aligned}$$

C. Flexible Architectural Design

The `SimpleMLP` architecture also allows multiple collections of hidden blocks to be stacked in sequence to create complex arrangements, such as funnels (ex. `[1*512, 2*256, 4*128, 8*64]`). To facilitate reproducible and large-scale experimentation, the architecture is defined via a compact string notation. For example, `--arch "[2*128, 4*64]"` specifies a network with two hidden blocks of 128 neurons, followed by four hidden blocks of 64 neurons.

However, for the purposes of our study's experimental design we limited ourselves to homogeneous network architectures such as `(1*2048)`, `(2*512)`, or `(8*8)`. We reserve research on heterogeneous architectures like `[2*128, 1*10, 4*16]` for our future work.

D. The "Frustration Engine"

The core of our methodology is a meta-learning loop that iteratively challenges the network to improve upon its historical best performance. We define the "Last Known Good" (LKG) as the model state with the highest-ever validation accuracy, and the "Bounty" as that accuracy score. A single meta-loop consists of:

1) **Clone LKG:** The cycle begins by cloning the weights of the current LKG model into a candidate model.
2) **Ablation:** PSA is applied by permanently ablating a single, randomly-chosen neuron in the candidate model. Ablation consists of zeroing the incoming weights, the bias, and, if applicable, outgoing weights.
3) **Train:** The newly damaged candidate model is trained for one full epoch.
4) **Evaluate:** If the new model's accuracy exceeds the current Bounty, it becomes the new LKG, and its score is the new Bounty. Otherwise, the candidate model is discarded.
5) **Repeat:** The process repeats from Step 1, ensuring improvements are only committed if the model successfully overcomes the damage.

Naturally, in the cases of the baseline training modes the ablative damage step is skipped, but otherwise the meta-loop proceeds in the same manner.

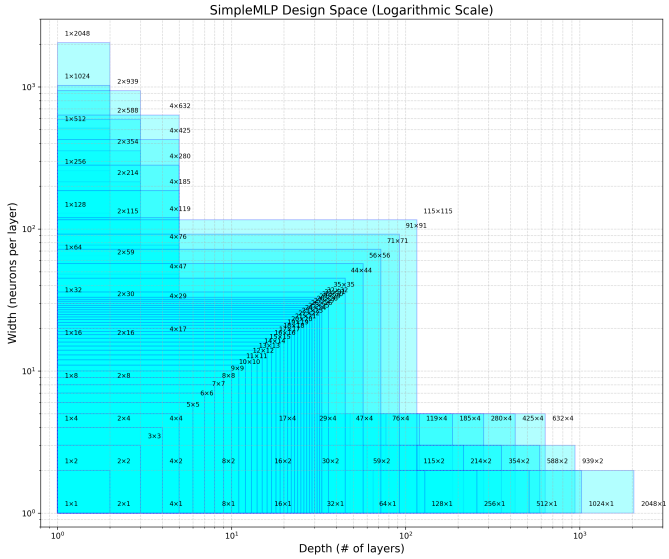

Fig. 3. The SimpleMLP architectural design space, visualized on a logarithmic scale. Width (neurons per layer) is plotted on the Y-axis, and depth (number of layers) is plotted on the X-axis. This visualization clearly demonstrates the exploration range from shallow-and-wide architectures (`1*2048`) to square architectures (`115*115`) and degenerate deep-and-narrow configurations (`2048*1`).

$$W_{\text{candidate}} = \mathcal{A}(W_{\text{LKG}}^{(t)})$$

$$W_{\text{challenger}} = \mathcal{T}(W_{\text{candidate}})$$

$$S_{\text{challenger}} = \mathcal{E}(W_{\text{challenger}})$$

$$W_{\text{LKG}}^{(t+1)} = \begin{cases} W_{\text{challenger}} & \text{if } S_{\text{challenger}} > S_{\text{LKG}}^{(t)} \quad \text{(Promote)} \\ W_{\text{LKG}}^{(t)} & \text{otherwise} \quad \text{(Discard)} \end{cases}$$

Fig. 4. The Persistent Stochastic Ablation (PSA) Meta-Learning Loop. The process iteratively corrupts the best-known state (*LKG*) and challenges the network to recover and surpass its previous performance benchmark before a new state is promoted.

E. Optimizer Resets

To ensure a fair and direct comparison between the proposed PSA methods and established baselines (e.g., `none`, `weight decay` and `dropout`), the optimizer state is reset at the beginning of each meta-loop. We felt allowing a persistent optimizer would be a confounding variable, as we only want to measure performance differences between the baseline/regularization and ablation strategies themselves, rather than benefit from any accumulated history within the optimizer's internal state (e.g., *momentum*). We believe the concept of a persistent optimizer learning from failure while routing around ablative damage to be an avenue for future research.

F. Deconstructing Training Effects

To understand the mechanisms of PSA, we designed six distinct training modes in order to disentangle possible effects, half in a baseline group (**None**, **Weight Decay**, **Dropout**), the other half in a group specifically dedicated to PSA techniques (**Full**, **Hidden**, **Output**).

- **none (Control):** The baseline model with no intervention.
- **decay (Weight Decay):** Standard weight decay applied with a fixed constant rate.
- **dropout (Dropout):** Standard dropout applied with a fixed constant rate.
- **full (Blended Pressure):** Randomly chooses a hidden linear layer from within the hidden blocks or the output layer, then chooses a random neuron to ablate. All incoming weights and the neuron's bias are zeroed (**partial ablation**). This represents a combination of the effects of the `hidden` and `output` modes.
- **hidden (Internal Pressure):** Randomly chooses a neuron from the list of all neurons available in the hidden linear layers from within the hidden blocks, then performs a **full ablation** on it, where the incoming weights, bias, and *outgoing* weights are zeroed.
- **output (External Pressure):** Randomly chooses a neuron from the linear output layer, then performs a *partial* ablation on it.

G. Isolating Training Effects

Due to the multiple training modes in our study, including **Weight Decay** and **Dropout**, we've taken great care to isolate the initialization parameters of the optimizer during training runs.

We chose `AdamW` [11] both for its standard support in `PyTorch` and because it decouples weight decay from the gradient update. This allows us to initialize the optimizer with `weight_decay = 0.0` for all modes except our specific `decay` mode, which sets it to `1e-4`. Thus, we have eliminated a possible confounding factor when testing the other modes, and followed the same philosophy we applied to separating the effects of **Dropout** as well.

Certain other initialization parameters are held constant across all modes:

- **Learning Rate (lr):** `1e-4`.
- **Betas (beta1, beta2):** `0.9`, `0.999`. Their momentum states are actually reset after each meta-loop.

We anticipate exploring compounding effects of Weight Decay and Dropout along with the other training modes in our future work.

A full list of training parameters is found in the Appendix: VI

H. Data Pipeline and Scalable Training

To ensure the rigor and reproducibility of our 5,880 trials, we implemented a data pipeline designed for statistical consistency, performance, and methodological soundness. These design decisions guarantee that any observed performance differences are attributable solely to the architectural and training strategy variations under experimentation.

1) Deterministic Dataset Splitting: We deterministically partitioned the standard MNIST dataset, comprising 60,000 training and 10,000 testing images, using a static seed (`1337`) to create a pseudo-random subset of 50,000 training images and 10,000 validation images from the original 60,000 image training set. The 10,000-image test set was reserved for unbiased, post-training performance evaluation, and was unseen by the models during the training process.

2) In-Memory Data Loading: Before training begins the entire MNIST dataset is loaded from disk once, converted into tensors, and stored in RAM. This eliminates the need for repetitive disk reads during the 100 meta-loops of each trial, greatly speeding up execution.

DataLoaders were configured with `num_workers=0` to avoid unnecessary multiprocessing overhead and `pin_memory=True` on CUDA-enabled systems.

3) AWS SageMaker AI: After validating the training process on multiple pieces of hardware, such as Apple Silicon (M3) and CUDA (NVIDIA A10G), we designed a set of project tools for utilizing Amazon Web Service's SageMaker AI to deploy hundreds of simultaneous training jobs.

We deployed 588 unique training jobs using `ml.g4dn.xlarge` instances, each tasked with running 100 meta-loops, storing the logs, writing a summary file, deleting the intermediary model, and starting over from scratch for each trial, for a total of 10 trial runs per job. No explicit pseudo-random seeding was assigned to the trials, relying instead on default hardware randomization processes.

We then collected and aggregated all the CloudWatch logs as well as `results.txt` summaries created by the SageMaker AI jobs, storing their data in the public repository as a reference for insights on the temporal dynamics of optimization during training and classification of validation results.

I. Automated Regime Classification

To ensure a reproducible analysis, we developed a sophisticated, rule-based algorithm to classify each of the 98 architectures into one of four distinct training regimes. The algorithm applies a sequence of cascading rules that operate on the summary statistics of the trial data for each architecture.

The algorithm's failure threshold, Z_{val}, is dynamically determined by calculating the majority class baseline (ZeroR) accuracy of the MNIST validation set (empirically derived as $\sim 11.02\%$).

Let M be the set of all training modes, with subsets B for baseline modes (`none`, `decay`, `dropout`) and A for ablative modes (`full`, `hidden`, `output`). For each mode $m \in M$, let μ_m, σ_m, and $P_m^{(\max)}$ be its mean, standard deviation, and peak accuracy, respectively. Let μ_B and σ_B be the mean and standard deviation of the set of baseline mean accuracies $\{\mu_b | b \in B\}$. We define an effective standard deviation, $\sigma_{eff} = \max(\sigma_B, 0.5\%)$, to establish a minimum tolerance for high-performance models with near-zero variance. The rules are applied in the following sequential order:

1) **Architectural Failure:** An architecture is classified as untrainable if it demonstrates a fundamental inability to learn. This is triggered if the peak performance of every single trial across all modes fails to exceed the ZeroR baseline.
$$\max_{m \in M}(P_m^{(\max)}) \leq Z_{val} \tag{4}$$

2) **Beneficial Regularization:** This regime identifies robust, over-parameterized models where all training strategies perform consistently well. An architecture is classified here only if it first passes a crucial check: no ablative mode can be a significant positive outlier. Specifically, if $\mu_a > (\mu_B + \sigma_{eff})$ for any $a \in A$, the model is disqualified from this regime. If it is not disqualified, it is classified as beneficial regularization if the means of all ablative modes fall within one effective standard deviation of the baseline mean.
$$\begin{aligned} &\text{if } \neg\exists a \in A \text{ s.t. } \mu_a > (\mu_B + \sigma_{eff}), \\ &\quad \text{then check if } |\mu_m - \mu_B| \leq \sigma_{eff} \quad \forall m \in M \end{aligned} \tag{5}$$

3) **Optimally Sized:** This regime characterizes models with limited redundant capacity, where interventions are generally harmful. An architecture is classified here if the mean performance of the ablative modes is significantly detrimental, falling more than one effective standard deviation below the mean of the baseline modes.
$$\mu_A < (\mu_B - \sigma_{eff}) \tag{6}$$

4) **Chaotic Optimization:** This regime is characterized by high instability and unpredictable performance, identified by any of the following three conditions: (a) *Optimizer Rescue:* All baseline modes consistently fail ($P_b^{(\max)} \leq Z_{val}$ for all $b \in B$) while at least one ablative mode achieves a successful learning outcome ($P_a^{(\max)} > Z_{val}$ for some $a \in A$). (b) *Stochastic Overachievement:* The ablative modes provide a significant average performance uplift over the baselines ($\mu_A > \mu_B + 0.5\%$). (c) *High Baseline Instability:* The performance of any baseline mode is extremely volatile, indicating chance-based success ($\sigma_b > 15\%$ for any $b \in B$).
5) **Unknown:** This isn't a regime as such; instead representing a failure of classification. None of the 98 configurations resulted in a failure to classify, and thus this can be used to further refine our regime analysis in the future should unclassifiable configurations be found.

This algorithm successfully classifies all 98 architectures, providing a robust and reproducible basis for the analysis visualized in Figure 5.

—

III. Results: The Four Regimes of Training (on SimpleMLP)

Our large-scale experiment, comprising 5,880 trials across 98 distinct MLP architectures, reveals that the efficacy of Persistent Stochastic Ablation is not universal but is instead governed by the interplay between a model's parametric capacity and its architectural stability, especially when compared to baseline techniques. We identified four distinct behavioral regimes, which are clearly demarcated in the architectural design space, as visualized in Figure 5.

A summary of the training results for each model, over 10 trials each lasting for 100 meta-loops, is presented in the Appendix: Table V.

A. Regime I: Beneficial Regularization in High Capacity, Stable Models

Located in the upper-left quadrant of the design space (Fig. 5, green points), this regime consists of shallow-and-wide architectures with high parametric capacity (e.g., `1*2048`, `2*939`, `4*632`). These models are sufficiently over-parameterized for the MNIST task, creating a landscape where nearly any reasonable training strategy can achieve high performance. We term this observed behavior **Beneficial Regularization**, not because every mode is a competitive regularizer, but because the models are robust enough to benefit from, or be unharmed by, various interventions.

The performance of our different training modes are tightly clustered in this regime. For the largest model, the `1*2048` architecture with `1,628,170` parameters, the **none** (Control) mode achieves the highest mean accuracy at **98.25%** and also captures the single-best trial performance at **98.37%**, suggesting that with extreme over-parameterization, the model has enough capacity to find an optimal solution without needing external regularization. However, the performance of `decay`, `dropout`, and PSA's `hidden` modes are statistically very close, demonstrating the model's resilience. As expected, the `output` ablation shows a comparatively deeper drop in performance, indicating that persistent damage to the logit layer can be detrimental even in a highly redundant network.

TABLE I
Performance Summary for the `1*2048` Architecture `1,628,170` Parameters (N=10).

Mode	Mean	Std	Min	Max
None	**98.25%**	0.06%	98.18%	**98.37%**
Decay	98.21%	0.04%	98.15%	98.26%
Dropout	98.20%	0.05%	98.10%	98.26%
Full	98.11%	0.07%	97.98%	98.21%
Hidden	98.19%	0.06%	98.07%	98.26%
Output	97.89%	0.12%	97.61%	98.10%

Our parameter-matching design allows us to observe the beneficial effects of principled regularization as these over-parameterized architectures gain depth. For the `2*939` architecture, **Dropout** emerges as the dominant strategy, securing the highest mean accuracy at **98.19%** and the highest peak performance at **98.29%**. This shift is further supported in the `4*632` architecture where `dropout` dominates again with a mean accuracy of **98.15%** and peak of **98.19%** , suggesting that Dropout becomes more effective in this regime as the number of available layers increases.

Interestingly, while traditional regularizers are superior on average, the stochastic nature of PSA allows it to occasionally discover outlier solutions. The **Hidden** ablation mode produced the highest peak accuracy for the `1*512` architecture at **98.10%**, narrowly exceeding the maximums of all other methods. This suggests that while less reliable, PSA's random exploration can, by chance, find configurations inaccessible to more stable optimizers.

This demonstrates a clear hierarchy where principled regularizers are most effective, but the random search of PSA provides a different, albeit less consistent, performance profile.

A promising direction for future work is to analyze the convergence dynamics over time; our preliminary results suggest that `dropout` converges most quickly (often in the first 50 meta-loops), followed by `decay` and `none`, while some PSA modes were still achieving performance improvements near the 100-meta-loop limit of our trials, hinting at a potentially longer but more exploratory optimization path.

B. Regime II: Optimal Sizing in Low Capacity, Stable Models

As model capacity decreases from the over-parameterized configurations, we enter a regime where the architectures are **Optimally Sized** for the MNIST task. These models (Fig. 5, red points), such as `1*128` or `4*119`, possess enough parameters to provide valuable solutions (e.g., with success rates $>$ 97%) even without the significant parametric redundancy of their much larger counterparts.

The defining characteristic of this regime is that any training intervention, whether principled regularization or persistent

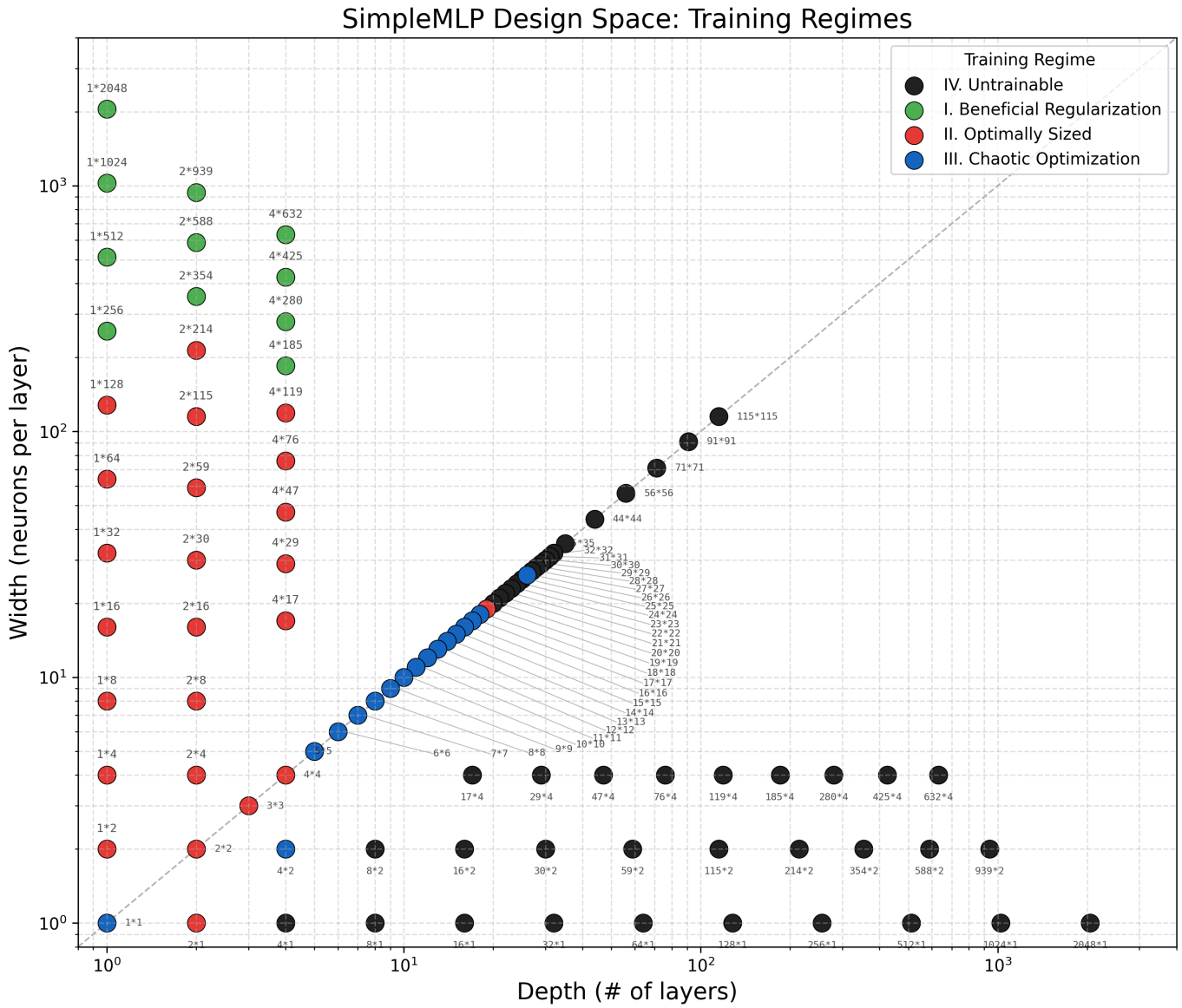

Fig. 5. The four distinct training regimes identified across the SimpleMLP architectural design space. Each point represents an architecture, plotted by its depth and width on a log-log scale. The color indicates the emergent behavioral regime, revealing a clear relationship between model shape, capacity, and the effectiveness of different training strategies. The dashed diagonal line ($y = x$) separates shallow-and-wide from deep-and-narrow configurations, and also represents the boundary of intentional parameter matching.

ablation, becomes increasingly detrimental when compared to the control. Further, the maximum performance of PSA techniques diverge quickly from their baseline regularizing counterparts, with their damage becoming catastrophic far more quickly than their peers.

In the `1*256` configuration with `203,530` parameters, sitting at the boundary between Beneficial Regularization and the Optimally Sizes regimes, the strongest PSA technique is `hidden` with a maximum of 97.81%, while the weakest baseline technique is `none` with a maximum of 97.83% for a divergence of $\sim$ 0.02%. Deeper into the regime, at the `1*64` configuration with `50,890` parameters, the maximums diverge significantly at $\sim$ 0.73% for `hidden` and `none`, at 96.15% and 96.86% respectively, illustrating the detrimental effects of PSA.

As shown in the "Winning Strategy Map" (Fig. 6), there is a clear shift away from `dropout`, first towards `decay` and then `none` as top performers when parameter counts drop towards less stable regimes. This indicates that the networks require their full parameter budget to function, and sensitivity to the preservation of weights is a defining feature when an architecture is already optimally sized.

The key insight from this regime is that PSA is not a universally applicable *regularizer*. Its effectiveness is contingent on the existence of network redundancy in simplistic MLP architectures. When that redundancy is unavailable in an

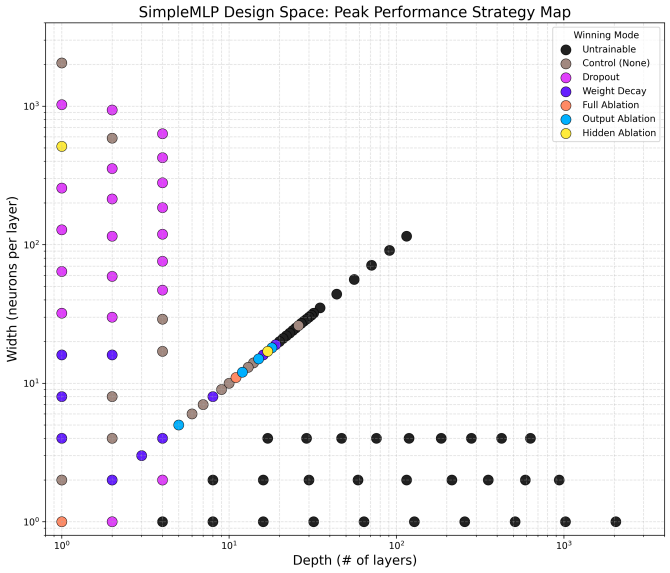

Fig. 6. Winning strategies across the SimpleMLP design space. Measured by the highest score from all trials per ablation type. Note that ablation techniques demonstrated utility in training degenerate or resource constrained network configurations. The singular "win" of the `hidden` ablation strategy in the `(18*18)` architecture stands out as an example of how persistent, random LKG ablation potentially acts as an escape mechanism from local optima.

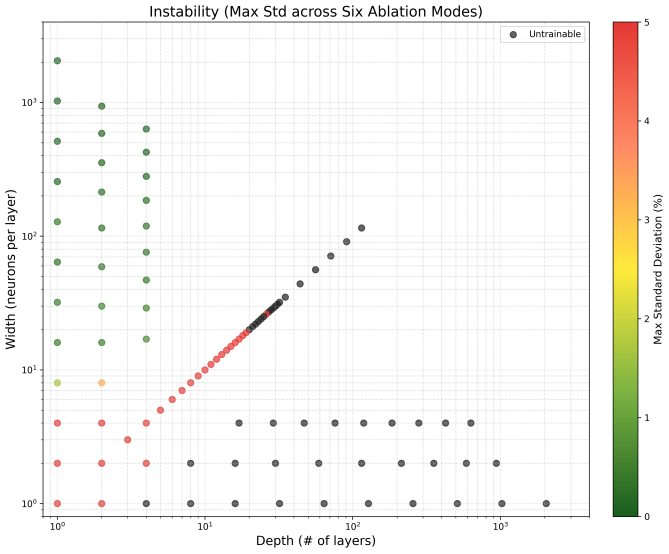

Fig. 7. Instability across the SimpleMLP design space, measured by the maximum standard deviation observed across the six training modes. The bright red diagonal band, corresponding to the Chaotic Optimization regime, highlights extreme performance variance, where PSA methods can induce high-scoring outliers even as baselines consistently fail. Black represents completely untrainable, architectural failures.

optimally sized model, PSA's "blind damage" is no longer a nudge towards a more robust solution but a direct impediment to the network's ability to learn.

This presents an interesting avenue of future research on its own: *detecting optimal network configurations* by comparing the baseline techniques to the rate of harm that the PSA techniques are able to inflict.

C. Regime III: Chaotic Optimization in Low Capacity, Unstable Models

The most surprising results of our study emerge in a regime defined by architectural instability and reduced parametric capacity. This region (Fig. 5, blue points) consists of models that are on the verge of the "Gradient Cliff", either too deep for back-propagation due to vanishing gradients (e.g., `18*18`), or so small that they lack any representational redundancy (e.g., `4*4`, `1*2`). Conventional gradient-based training fails in these pathological configurations, but the violent, non-gradient-based intervention of PSA seems to become a strikingly effective - albeit chaotic - optimization method.

The defining characteristic of this regime is a dramatic divergence in "uplifting" behavior of ablative over baseline methods. As shown in the "Baseline Performance" map (Fig. 8), the `none` (Control) and other traditional regularization models in this area consistently fail, yielding accuracies near the ZeroR of the validation dataset ($\sim$11.02%); trapped in a poor local minima converged on simply guessing the validation dataset's single largest MNIST class, a state that `dropout` and `decay` are demonstrably unable to rescue them from.

In positive contrast, the "Ablation Effects" map (Fig. 9) reveals this same region as the only area where PSA is

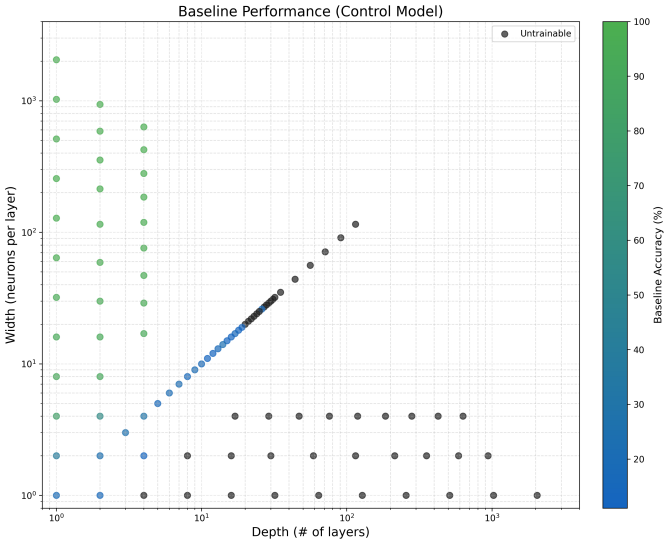

Fig. 8. Baseline performance heatmap across the SimpleMLP design space. Architectures with depth $\geq \sim$27 or utilizing degenerately narrow widths are untrainable across all ablation modes. These boundaries are a manifestation of the Vanishing Gradient Problem.

consistently beneficial. (green). The "Instability" map (Fig. 7) provides valuable insight even in the over-parameterized regime: while many PSA trials may fail, some achieve measurably superior successful outcomes. This supports our hypothesis that PSA acts as a /textbfstochastic kick, allowing the optimizer to escape traditionally difficult or intractable local optima.

A powerful example is the `18*18` architecture with `20,134` parameters. Table II shows a total failure of the

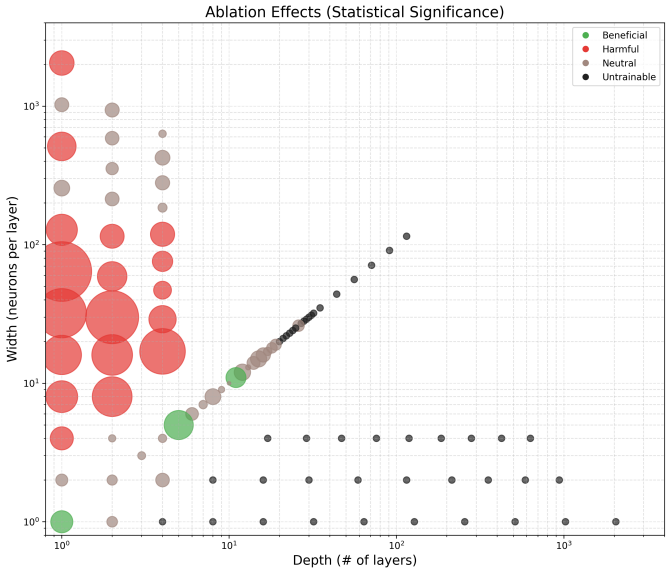

Fig. 9. Ablation effects (beneficial or harmful) across the SimpleMLP design space. In the trainable models there are clear regimes where ablation either is beneficial, harmful, or indistinguishable from the baseline.

baseline methods (e.g., a maximum of 11.02% over all 10 trials). However, the `full` and `output` PSA modes produce demonstrable improvements in peak performance, at 19.29% and 71.16% respectively.

TABLE II
PERFORMANCE SUMMARY FOR THE `18*18` ARCHITECTURE `20,134` PARAMETERS (N=10).

Mode	Mean	Std	Min	Max
None	10.48%	0.51%	9.81%	11.02%
Decay	10.22%	0.74%	8.65%	11.02%
Dropout	10.18%	0.39%	9.73%	10.92%
Full	**11.20%**	2.74%	9.81%	**19.29%**
Hidden	10.11%	0.47%	9.73%	11.02%
Output	**18.87%**	17.87%	10.12%	**71.16%**

Critically, the separation of PSA into three different modes allows us to disentangle the effects of ablative placement and method. The "Winning Strategy Map" (Fig. 6) shows that no single PSA mode dominates this chaotic regime; instead, the optimal strategy depends on the specific architectural pathology.

- **Hidden Ablation (Full Neuron Ablation):** For the deep but unstable square model `17*17`, **Hidden** ablation was the most effective strategy. This method performs a *full neuron ablation* by zeroing both the bias, incoming, and outgoing weights of randomly chosen hidden neurons. It suggests that for networks trapped by the Vanishing Gradient Problem, a more aggressive stochastic "kick" may fundamentally alter the network's functional pathways and escape the poor local minimum.
- **Full & Output Ablation (Partial Ablations):** In contrast, for extremely small, resource-starved models such as `1*1` and `5*5`, the winning strategies were **Full** and **Output** ablation. These methods use a less destructive *partial ablation* (zeroing only incoming weights and the bias). This finding suggests that when a model has very few parameters to begin with, a more targeted perturbation is more effective. Note that the `output` is the most aggressive and chaotic of the three modes due to directly zeroing the weights of output logits.

This regime provides the strongest evidence for the non-regularization utility of PSA. It demonstrates that blind, persistent damage can serve as a powerful tool for online stochastic optimization, offering a lifeline to models that might be otherwise untrainable with standard gradient-based methods.

Our findings around the "stochastic kick" behavior, where a random perturbation might dramatically improve an otherwise failing training optimization process, seems conceptually similar to the effectiveness of random search for hyperparameter optimization, often outperforming grid search in high-dimensional, non-convex problem spaces [7].

This reasoning is further supported by the success of `none`, `decay`, and `dropout` in the `19*19` architecture where each, by chance in 1 trial out of 10, randomly discovered a viable above-ZeroR optimization path that allowed the model to learn despite deepening failure; the same logic extends to the unexpected trainability of the `26*26` architecture with `none` and `output`. This demonstrates the need for even further trials (ex. 100 for each mode) to fully map the probabilistic boundaries of the Vanishing Gradient Problem in `SimpleMLP`.

D. Regime IV: Architectural Failure in Zero Capacity, Unstable Models

The final regime encompasses architectures that are *fundamentally untrainable*: where the choice of training strategy becomes irrelevant. Our systematic exploration of the design space was designed to map the precise boundary of this failure as an observable phenomenon where even the powerful stochastic kick of PSA is insufficient to rescue the model from architectural collapse.

Models in this regime (Fig. **??**, black points) universally fail to learn, with all six training modes stalling at an accuracy of approximately 11.02%. This value is the empirically measured majority class baseline (ZeroR) for our 10,000-sample validation set, confirming a total failure to learn from input features. This failure is attributable to two distinct architectural pathologies: the *Vanishing Gradient Problem* in excessively deep networks, and *Information Bottlenecks* in degenerately narrow ones.

1) The VGP Boundary in Square Architectures: One of the primary goals of our study was to identify the exact depth at which a simple MLP becomes untrainable on MNIST. Our exploration of square models revealed this boundary between a depth of 19 and 20 layers.

As shown in Table III, the `19*19` architecture resides firmly within the Chaotic Optimization regime. While the majority of modes fail - including the PSA variants that were successful in

18*18 - the **Decay** mode is able to randomly find a trainable path, achieving a remarkable peak accuracy of 79.87% in a single trial. This demonstrates that at a depth of 19, the model is on the absolute edge of failure yet remains salvageable through specific, albeit chaotic, training dynamics.

The 20*20 architecture, just one layer deeper, *completely fails to train*. All six training modes, including every PSA variant, fail completely, with no trial exceeding the 11.02% ZeroR. Neither lucky starting conditions for the baseline regularization techniques, nor are the aggressive stochastic kicks of PSA powerful enough to escape a loss landscape that has become intractably flat due to the compounded decay of gradients through 20 successive layers.

TABLE III
PERFORMANCE COLLAPSE AT THE VGP BOUNDARY (N=10). THE 19*19 ARCHITECTURE REMAINS SALVAGEABLE BY CHANCE (UTILIZING DECAY), WHILE THE 20*20 ARCHITECTURE SHOWS COMPLETE TRAINING FAILURE ACROSS ALL MODES.

	19*19 (Chaotic but Salvageable)				20*20 (Untrainable)			
Mode	**Mean**	**Std**	**Min**	**Max**	**Mean**	**Std**	**Min**	**Max**
None	11.34%	3.02%	8.65%	20.12%	10.13%	0.45%	9.72%	11.02%
Decay	**17.24%**	20.88%	9.73%	**79.87%**	10.69%	0.48%	9.72%	11.02%
Dropout	11.50%	2.98%	9.72%	20.30%	10.19%	0.54%	9.72%	11.02%
Full	10.46%	0.55%	9.73%	11.02%	10.59%	0.74%	8.65%	11.02%
Hidden	10.46%	0.58%	9.72%	11.02%	10.52%	0.51%	9.72%	11.02%
Output	10.61%	0.48%	9.81%	11.02%	11.02%	0.00%	11.02%	11.02%

However, further results also reveal that this boundary is not a simple, binary cut-off, but rather a probabilistic one. For the much deeper 26*26 architecture, whose neighbors were otherwise untrainable, a single trial of the none (Control) mode achieved a peak accuracy of 73.34%. This outlier success suggests that viable optimization paths still exist even as the VGP becomes overwhelming, but are so narrow that they are only found by chance. This particular finding demonstrates that the ten trials conducted per configuration in this study were insufficient to fully map the probabilistic influence of the VGP in deeper topologies.

Still, identification of this boundary highlights the limitations of our SimpleMLP testbed. Architectures like ResNet [8] or the more recent ResMLP [9] were designed specifically to overcome this boundary by introducing skip connections, providing uninterrupted paths for gradient flow, and enabling the training of much deeper networks.

2) Information Bottlenecks in Deep-and-Narrow Architectures: The second cause of architectural failure is the information bottleneck. This occurs in our "deep-and-narrow" configurations (e.g., L*4, L*2, L*1). While the shallowest of these, such as 1*1 and 4*2, are technically trainable (albeit with poor performance, landing them in the Chaotic Optimization regime), they quickly become untrainable as depth increases.

It is self-evident that collapsing the 784-pixel input of the MNIST dataset into a 1-neuron wide set of layers irrevocably destroys critical spatial data, and these maximally degenerate cases were included not just for the sake of "completeness" but to identify the boundaries where any possible learning *stops*, especially when compared to other deep-and-narrow yet slightly wider topologies.

E. Empirical Convergence Dynamics

By examining the detailed CloudWatch logs from our wide-scale run of 5,880 trials we extracted the temporal pattern of the validation accuracy's improvement, stalling, and failures for each of the 588 experimental configurations across all 10 of their 100 meta-loop trial runs.

While final peak accuracy may reveal the ultimate effectiveness of a given training strategy, the temporal pattern of each trial offers deep insights into the underlying optimization dynamics, allowing us to directly observe the stability and convergence properties of each combination of architecture and training technique. This detailed data, visualized for several key architectures in Figure 10, provides clear evidence for the different roles played by baseline and PSA methods and empirical support for our reasoning about classification into four training regimes.

1) Stable Convergence vs. Exploratory Volatility: In stable architectures, such as the over-parameterized 1*2048 model (Figure 10, Top), all six training modes exhibit smooth and rapid convergence. The baseline methods (none, decay, dropout) quickly achieve a high-performance plateau. The PSA modes also converge to a similar performance ceiling over a longer time horizon, still gaining minor improvements even up to the final meta-loops, demonstrating both the inherent redundancy of the larger models allowing consistent recovery from the stochastic ablation, and a need to do further trials beyond 100 meta-loops to measure how far continued PSA-driven improvements might extend during training.

The dynamics diverge dramatically in the **Chaotic Optimization** regime. For the 18*18 architecture (Figure 10, Second from Bottom), the baseline methods are completely unable to escape a quickly identified, poor local minimum: flat-lining at the ZeroR baseline for all 100 meta-loops. The learning curves for the PSA methods, however, are extremely volatile. This is the direct empirical signature of the "Frustration Engine" at work: each point represents a recovery attempt from a newly damaged state. Many attempts fail, resulting in performance drops back towards the baseline. However, this high-variance search process seems to allow the optimizer to explore a much wider solution space. We believe it is this chaotic exploration that enables PSA to discover optimization pathways that lead to significant performance breakthroughs, reaching accuracies far beyond what the stable, gradient-based methods achieve on their own.

2) Cost and Benefit of Late-Stage Breakthroughs: This volatility represents the inherent "cost of exploration" in the PSA paradigm, as many candidate models are ultimately discarded. Our analysis of the LKG growth data, however, reveals that this is the central mechanism of the method's utility in unstable architectures. The repeated "kicks" from a disadvantaged state seem to prevent the optimizer from getting permanently trapped.

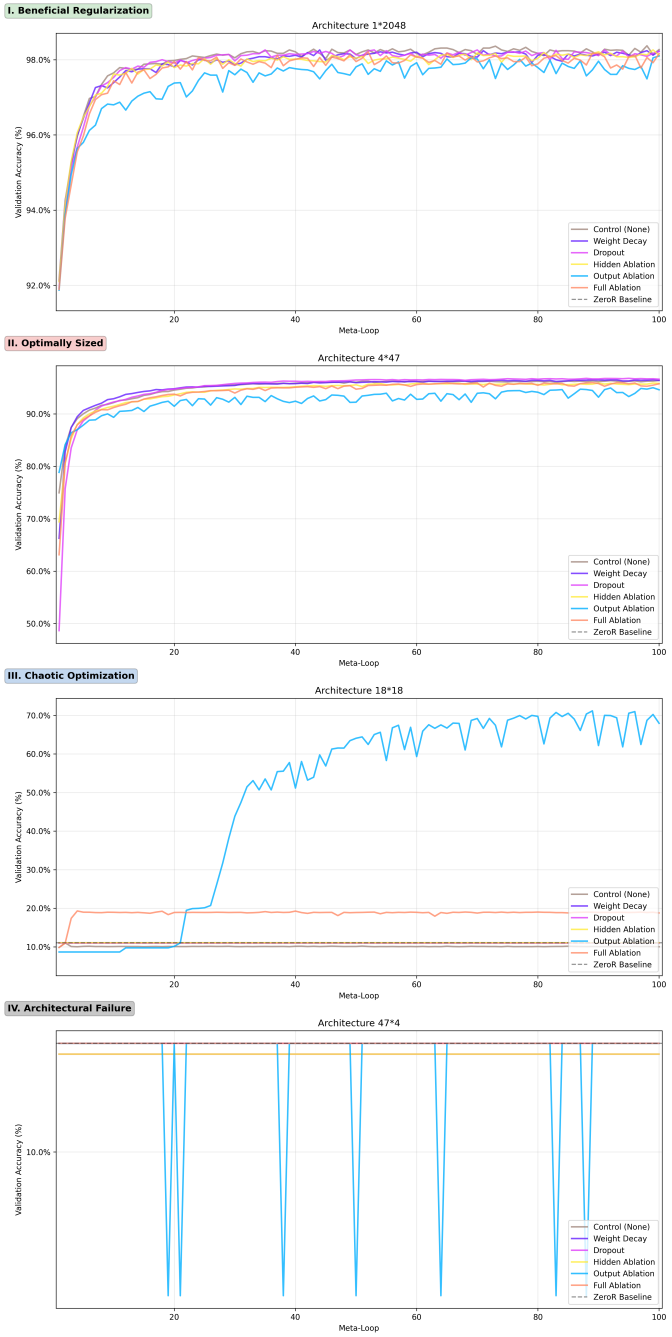

Fig. 10. Convergence dynamics across training regimes, plotting per-"meta-loop" validation accuracy for all 10 trials of each mode, with the peak performing trials for a given architecture plotted.

The LKG growth data provides multiple instances of these late-stage breakthroughs, showing significant improvements from PSA modes after baseline methods have plateaued (Fig. 11):

- **`18*18` (Output):** This is a quintessential example of "optimizer rescue." While all baseline modes stall at the ZeroR baseline, the `output` mode "ignites" after four dormant meta-loops with a massive **10.45%** improvement at *meta-loop 5*. This begins an exploratory search for the rest of the run, achieving another significant improvement of **0.77%** as late as *meta-loop 89*.
- **`17*17` (Hidden):** This architecture showcases PSA as a truly **sustained** exploratory search. Rather than a single breakthrough, the `hidden` mode consistently finds new paths to improvement throughout the entire run, including gains at *meta-loop 79* (**1.31%**), *meta-loop 92* (**0.45%**), and even on the *100th and final meta-loop* (**0.20%**); an elegant demonstration that the search space wasn't

exhausted.

- `10*10` **(Full):** This case demonstrates an explosive, late-stage learning breakthrough. After a period of inconsistent gains, the `full` mode began a dramatic and sustained cascade of improvements starting at *meta-loop 86*. In the final 15 loops of the trial, the model gained over **24%** in absolute validation accuracy, showing that PSA can unlock periods of rapid optimization long after initial learning has slowed.
- `4*2` **(Decay & Dropout):** This degenerately small architecture demonstrates that even baseline methods can exhibit late-stage exploration under extreme constraints. The `decay` trial stagnated over a long period before its last sudden, significant improvement at *meta-loop 96* (**0.12%**). Dropout, on the other hand, showed small but continuous improves all the way to the /textit100th meta-loop, suggesting that for certain unstable configurations premature convergence can be prevented by the inherent stochasticity of the training mode.

The success of PSA in the Chaotic Optimization regime demonstrates that for certain pathological landscapes, switching from monotonic convergence in favor of a volatile, persistent search is a highly effective, albeit potentially lengthy, optimization strategy.

It also leaves the door open for post-regularization "kicks" into nearby but valuable optima once a traditional regularization technique like Dropout has done all it can, and this will be an avenue of further research.

IV. Discussion and Limitations

The four regimes identified in our results provide a unified theory for the behavior of PSA on simplistic MLPs. Its utility is an emergent property represented within a two dimensional phase space of optimization opportunities across the non-linear axes of parametric capacity and architectural stability. The "gradient cliff" encountered in Regime IV, however, places a hard limitation on this initial study. We cannot know how PSA affects truly deep networks if the baseline architecture itself is untrainable.

Our exhaustive study reveals that the primary utility of Persistent Stochastic Ablation is not as a regularizer, but as a surprisingly powerful tool for stochastic optimization. The most significant finding of this work is the emergence of the "Chaotic Optimization" regime, where the random, non-gradient-based "kick" from PSA provides a lifeline to architecturally unstable models.

For these pathological networks, trapped in poor local minima by the Vanishing Gradient Problem or degenerate Information Bottlenecks, PSA provides a method capable of inducing meaningful learning, outperforming conventional baselines like Dropout and Weight Decay. While PSA can confer a minor regularizing effect in stable, over-parameterized models, this benefit is neither statistically significant nor competitive with established techniques over the training intervals we observed (100 meta-loops).

We find that PSA's currently demonstable value lies not in refining well-behaved models, but in rescuing otherwise untrainable ones, possibly positioning PSA as a novel mechanism for exploring difficult or intractable loss landscapes and motivating our future work in applying this paradigm to more complex architectures where optimization challenges persist.

Key limitations of this initial study include:

- **Architectural Weakness:** The study was confined to simple MLPs, which lack modern components like residual connections (as in ResNet) [8] or normalization layers (like Batch Norm) [10]. The observed efficacy of PSA might be entirely different in more complex and robust architectures, or disappear altogether.
- **Single, Simple Dataset:** All experiments were conducted on MNIST. While useful for a proof-of-concept, its low complexity is not representative of the challenges posed by more complex, real-world datasets (e.g., CIFAR-100, ImageNet). The benefits of PSA could be magnified, diminished, or altered on more difficult problems.
- **Fixed Ablation Strategy:** The PSA mechanism was fixed to ablating exactly one neuron per meta-loop. The effects could change dramatically with different strategies, such as ablating a percentage of neurons, targeting specific layers, or adaptively varying the rate of ablation over the course of training.
- **Fixed Optimizer Hyperparameters:** The underlying optimization algorithm (e.g., AdamW) and its hyperparameters (e.g., learning rate) were held constant across the vastly different architectures. It is possible that some of the "Architectural Failures" could have been mitigated with architecture-specific tuning, potentially altering the winning ablation mode in some regimes.
- **Limited Training Intervals:** We trained each model for only 100 meta-loops, and while existing regularization techniques like Dropout and Weight Decay quickly converged with the first 30-75% of the training intervals, several PSA modes were still finding improvements into the 99-th meta-loop, suggesting that the reported peak accuracies for PSA may be understated and that longer training runs are an important avenue for future work.
- **Greedy Advancement Mechanism:** The initial "Frustration Engine" is greedy, only accepting a new model if its performance is strictly better. This could prevent the training process from taking a temporary step back in accuracy to navigate a ridge in the loss landscape that might lead to a much better eventual solution. We will address this in our future work by implementing a patience mechanic into the Frustration Engine, allowing it a certain number of retries before accepting a new LKG from an inferior model if the Bounty is not met.

V. Conclusion and Future Work

This study successfully characterized the complex, context-dependent behavior of Persistent Stochastic Ablation (PSA) and whether it could act as a beneficial "evolutionary pressure" on simplistic MLP architectures. The comprehensive statistical

analysis of 5,880 independent trials reveals a clear but multi-faceted answer: the utility of PSA is an emergent property governed by a model's parametric capacity and stability. Our findings definitively characterize three distinct, statistically significant regimes of behavior, while allocating a fourth regime to mark untrainable networks due to their inherent architectural limitations.

First, in **over-parameterized**, trainable networks, the persistent, principled pressure of randomized hidden neuron ablation acts as an exploratory mechanism. While producing a mean accuracy statistically indistinguishable from the control, it in some cases increased the performance ceiling, enabling the discovery of higher-scoring model states than the optimizer found on its own.

Second, in **well-sized to constrained (but trainable)** networks, all forms of ablation were detrimental to mean performance. However, the data reveals a clear hierarchy of damage: the hidden ablation is consistently and significantly less harmful than the maximally aggressive, targeted output ablation - despite the former being able to fully ablate its incoming /textitand outgoing weights. This demonstrates both the value of disentangling the ablation modes via separate `hidden`, `output`, and `full` modes.

Third, and most surprising, in architecturally flawed, degenerately designed, or critically under-powered networks where standard gradient-based optimization fails, PSA provides a demonstrably effective mechanism for escaping poor local optima - especially the `output` ablation technique which targets the final linear layer of outputs. Where the baselines failed or consistently under-performed in these architectures, the ablation modes repeatedly "kicked" the optimizer out of gradient training traps. This provides yet another direction of research to explore in our upcoming work.

Finally, the definitive training failures of deeper `SimpleMLP` models motivates the clear next step: to re-implement this experimental harness using a **ResMLP-based** architecture [9]. The use of residual connections should overcome the vanishing gradient problem, allowing us to investigate the true effect of PSA in networks that are both extremely deep and fully trainable. This will be the focus of our next study.

Overall, we have shown that Persistent Stochastic Ablation is a promising research direction. It not only offers a demonstrably novel mechanism for escaping local optima in degenerate or otherwise challenging network configurations, but a possible path towards more robustly trainable AI models.

1) Future Work:

- **Implementation of ResMLP:** In order to explicitly explore and measure the effects of PSA in deeper architectures we will reimplement the experiments using a ResMLP testbed while replicating the original paramters of our study. This will clarify any of our results that were completely dependent on the instability of the underlying test architectures, and help measure further limits of the Vanishing Gradient Problem and Information Bottlenecks in degenerate topologies.
- **Additional Trials testing the VGP:** The `26*26` architecture makes it clear that we have not fully understood the behavior of the Vanishing Gradient Problem as a probablistic impact on optimization paths, motivating a follow-up study with a greatly increased number of trials (e.g., n=100), focused specifically on the square architectures (`1*1` to `32*32`).
- **Convergence dynamics analysis over time:** Preliminary results suggest different convergence patterns between methods (dropout converging quickly vs. PSA showing continued improvement near the 100-meta-loop limit), warranting systematic investigation of temporal training dynamics.
- **Disrupting training traps:** Because the SimpleMLP hidden blocks share the same structure for all modes we are able to train partially in one mode and then resume training in another. This means we can explore initiating "stochastic kicks" even when Dropout has already converged, and vice versa.
- **Frustration with Patience:** We propose an update to our Frustration Engine's meta-loops that would allow a "patience" dynamic, allowing geometric "retries" and accepting lower performing models as subsequent LKG states after exhausting the local attempts, while retaining the Bounty as a method to reset the number of required retries when a truly improved model is found.
- **Cross-dataset validation beyond MNIST:** This initial study's limitation to MNIST leaves open questions about PSA's effectiveness on more complex datasets that we wish to test.
- **Heterogeneous network architectures:** We intend to also research complex configurations, such as funnel-like architectures (e.g., [2*128, 1x10, 4x16]), which could reveal how PSA performs when applied to networks with varying layer capacities and information flow patterns.
- **Persistent optimizer state across meta-loops:** The concept of allowing optimizers to maintain momentum and other internal states while learning to route around ablative damage represents an unexplored avenue that could significantly enhance PSA's effectiveness.
- **Detecting optimal network configurations using PSA harm rates:** Our observation that PSA's rate of harm correlates with optimization optimality suggests a novel method for automatically determining appropriate model capacity for tasks.
- **Random search parallels in high-dimensional optimization:** The similarity between PSA's stochastic kick behavior and the effectiveness of random search in hyper-parameter optimization suggests broader applications in non-convex optimization landscapes, and is also worthy of exploration for providing a grounded, mathematical understanding into how and why PSA works.
- **Varied online ablation strategies:** Stochastic ablation was an obvious first research avenue, but we also wish to explore persistent fixed ablation, especially in output layers. Additionally, researching intelligent strategies that

react to network conditions is also of high interest.

ACKNOWLEDGMENT

This project was developed with the assistance of several large language models (LLMs), which served as interactive tools to accelerate the research workflow. Their specific contributions included conceptualization, code generation, debugging, and documentation. The primary models consulted were Google's Gemini Pro, OpenAI's GPT-4, Anthropic's Claude, and xAI's Grok. While these tools were integral to the development process, the intellectual direction, experimental design, and all final conclusions are the sole work of the human author.

REFERENCES

[1] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 598–605.

[2] B. Hassibi and D. G. Stork, "Second Order Derivatives for Network Pruning: Optimal Brain Surgeon," in *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 164–171.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[4] G. M. Edelman, *Neural Darwinism: The Theory of Neuronal Group Selection*. New York, NY, USA: Basic Books, 1987.

[5] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 2, pp. 107–116, 1998.

[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 1126–1135, Aug. 2017.

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[9] H. Touvron et al., "ResMLP: Feedforward Networks for Image Classification With Data-Efficient Training," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 5314–5321, April 2023, doi: 10.1109/TPAMI.2022.3206148.

[10] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[11] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

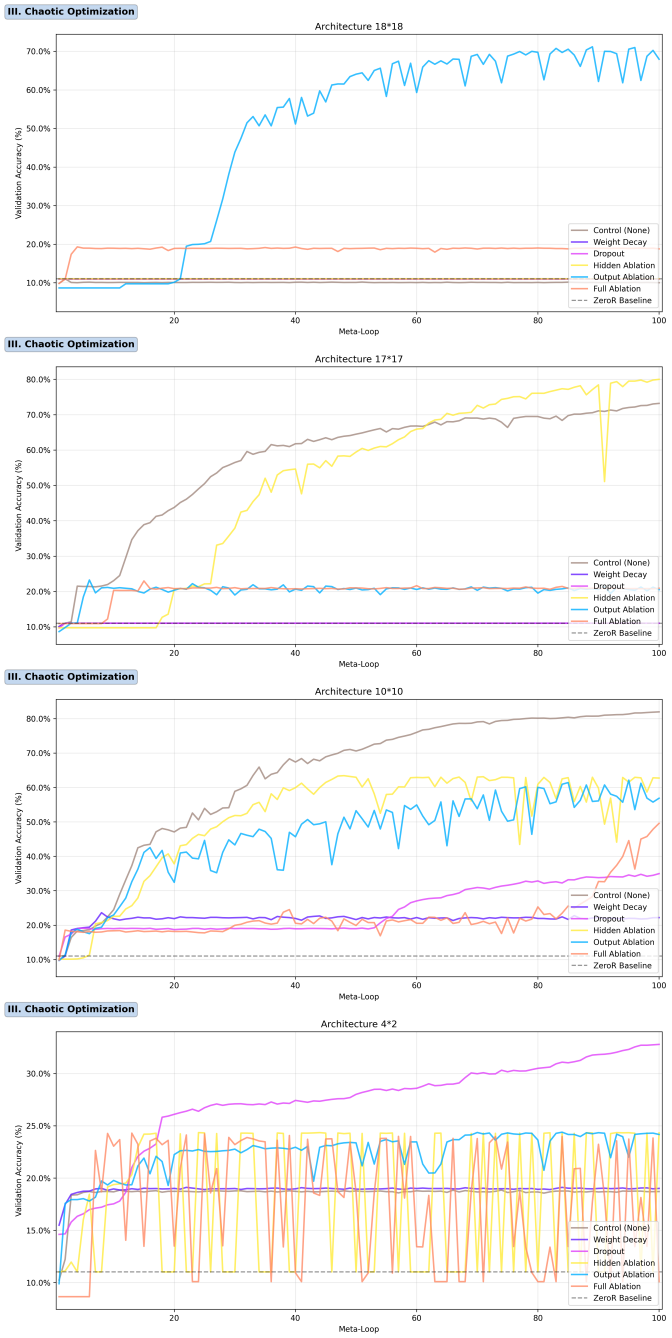

Fig. 11. Late-stage convergence dynamics for four architectures in the Chaotic Optimization regime. Each line plots the peak validation accuracy (LKG) of a single, best-performing trial out of ten for each mode over 100 meta-loops. These plots clearly illustrate PSA's ability to **rescue learning** in otherwise intractable landscapes (e.g., `18*18`), where baseline methods stall at the ZeroR; unlock an **explosive cascade** of late-stage improvements (`10*10`); and act as a **sustained exploratory mechanism** until the final training loops (`17*17`). Finally, the pathologically small `4*2` architecture reveals that even baseline methods like `decay` and `dropout` can exhibit late-stage exploration under extreme constraints.

APPENDIX

This appendix provides the supplementary data referenced in the main text, including the complete list of tested architectures, results across modalities, and the hyperparameters used for all training runs.

TABLE IV
CONFIGURATIONS & PARAMETERS FOR NETWORK ARCHITECTURES

Architecture	Parameters	Parameter Matching
1*2048	1,628,170	Matched
1*1024	814,090	Matched
1*512	407,050	Matched
1*256	203,530	Matched
1*128	101,770	Matched
1*64	50,890	Matched
1*32	25,450	Matched
1*16	12,730	Matched
1*8	6,370	Matched
1*4	3,190	Matched
1*2	1,600	Matched
2*939	1,629,175	Matched
2*588	813,802	Matched
2*354	407,110	Matched
2*214	216,150	Matched
2*115	104,775	Matched
2*59	50,455	Matched
2*30	24,790	Matched
2*16	13,002	Matched
2*8	6,442	Matched
2*4	3,210	Matched
4*632	1,702,618	Matched
4*425	881,035	Matched
4*280	458,650	Matched
4*185	250,315	Matched
4*119	137,455	Matched
4*76	77,986	Matched
4*47	44,143	Matched
4*29	25,675	Matched
4*17	14,443	Matched
115*115	1,612,195	Matched
91*91	825,835	Matched
71*71	414,295	Matched
56*56	220,090	Matched
44*44	120,130	Matched
35*35	70,675	Matched
32*32	58,186	Matched
31*31	54,415	Matched
30*30	50,830	Matched
29*29	47,425	Matched
28*28	44,194	Matched
27*27	41,131	Matched
26*26	38,230	Matched
25*25	35,485	Matched
24*24	32,890	Matched
23*23	30,439	Matched
22*22	28,126	Matched
21*21	25,945	Matched
20*20	23,890	Matched
19*19	21,955	Matched
18*18	20,134	Matched
17*17	18,421	Matched
16*16	16,810	Matched
15*15	15,295	Matched
14*14	13,870	Matched
13*13	12,529	Matched
12*12	11,266	Matched
11*11	10,075	Matched
10*10	8,950	Matched
9*9	7,885	Matched
8*8	6,874	Matched
7*7	5,911	Matched
6*6	4,990	Matched
5*5	4,105	Matched
4*4	3,250	Matched
3*3	2,419	Matched
2*2	1,606	Matched
1*1	805	Matched
632*4	15,810	Asymmetric
425*4	11,670	Asymmetric
280*4	8,770	Asymmetric
185*4	6,870	Asymmetric
119*4	5,550	Asymmetric
76*4	4,690	Asymmetric
47*4	4,110	Asymmetric
29*4	3,750	Asymmetric
17*4	3,510	Asymmetric
939*2	7,228	Asymmetric
588*2	5,122	Asymmetric
354*2	3,718	Asymmetric
214*2	2,878	Asymmetric
115*2	2,284	Asymmetric
59*2	1,948	Asymmetric
30*2	1,774	Asymmetric
16*2	1,690	Asymmetric
8*2	1,642	Asymmetric
4*2	1,618	Asymmetric
2048*1	4,899	Asymmetric
1024*1	2,851	Asymmetric
512*1	1,827	Asymmetric
256*1	1,315	Asymmetric
128*1	1,059	Asymmetric
64*1	931	Asymmetric
32*1	867	Asymmetric
16*1	835	Asymmetric
8*1	819	Asymmetric
4*1	811	Asymmetric
2*1	807	Asymmetric

TABLE V: Mean of Validation Peak Accuracy (%) with Standard Deviation over 10 Trials of 100 Meta-Loops for SimpleMLP Architectures

Architecture	none Mean ± Std (%)	decay Mean ± Std (%)	dropout Mean ± Std (%)	full Mean ± Std (%)	hidden Mean ± Std (%)	output Mean ± Std (%)	Winner
$1 * 2048$	98.25 ± 0.06	98.21 ± 0.04	98.20 ± 0.05	98.11 ± 0.07	98.19 ± 0.06	97.89 ± 0.12	none
$1 * 1024$	98.09 ± 0.05	98.12 ± 0.03	98.11 ± 0.07	98.05 ± 0.06	98.10 ± 0.08	97.71 ± 0.12	decay
$1 * 512$	98.01 ± 0.05	97.92 ± 0.09	97.95 ± 0.11	97.85 ± 0.09	97.95 ± 0.07	97.35 ± 0.11	none
$1 * 256$	97.62 ± 0.19	97.71 ± 0.07	97.82 ± 0.08	97.47 ± 0.18	97.69 ± 0.07	96.65 ± 0.19	dropout
$1 * 128$	97.37 ± 0.14	97.41 ± 0.14	97.53 ± 0.13	96.78 ± 0.18	97.10 ± 0.09	95.81 ± 0.41	dropout
$1 * 64$	96.74 ± 0.12	96.78 ± 0.16	96.98 ± 0.14	95.46 ± 0.14	95.81 ± 0.23	94.65 ± 0.29	dropout
$1 * 32$	95.49 ± 0.20	95.47 ± 0.21	95.61 ± 0.20	93.04 ± 0.32	93.80 ± 0.41	93.00 ± 0.49	dropout
$1 * 16$	93.12 ± 0.33	93.42 ± 0.47	93.41 ± 0.30	91.33 ± 0.34	91.82 ± 0.18	91.43 ± 0.41	decay
$1 * 8$	91.14 ± 0.24	91.24 ± 0.33	90.30 ± 0.44	86.44 ± 1.70	86.58 ± 1.67	89.07 ± 0.45	decay
$1 * 4$	81.35 ± 3.53	83.56 ± 1.13	80.25 ± 2.16	67.04 ± 9.28	60.16 ± 9.26	72.75 ± 3.58	decay
$1 * 2$	40.28 ± 15.04	46.90 ± 12.69	40.37 ± 10.62	33.50 ± 10.07	20.19 ± 2.83	36.58 ± 7.41	decay
$2 * 939$	98.09 ± 0.05	98.15 ± 0.04	98.19 ± 0.06	98.13 ± 0.05	98.17 ± 0.05	97.96 ± 0.08	dropout
$2 * 588$	98.05 ± 0.08	98.04 ± 0.06	98.13 ± 0.06	98.00 ± 0.08	98.06 ± 0.07	97.87 ± 0.07	dropout
$2 * 354$	97.94 ± 0.12	97.95 ± 0.08	98.11 ± 0.08	97.89 ± 0.08	97.91 ± 0.11	97.54 ± 0.13	dropout
$2 * 214$	97.68 ± 0.14	97.68 ± 0.09	97.91 ± 0.09	97.60 ± 0.12	97.69 ± 0.07	97.01 ± 0.21	dropout
$2 * 115$	97.32 ± 0.10	97.34 ± 0.18	97.58 ± 0.18	97.09 ± 0.13	97.27 ± 0.13	96.07 ± 0.30	dropout
$2 * 59$	96.65 ± 0.18	96.62 ± 0.10	96.94 ± 0.19	95.84 ± 0.32	96.01 ± 0.17	94.63 ± 0.32	dropout
$2 * 30$	95.44 ± 0.20	95.38 ± 0.27	95.55 ± 0.29	93.48 ± 0.29	93.77 ± 0.52	93.10 ± 0.51	dropout
$2 * 16$	93.49 ± 0.32	93.65 ± 0.49	92.95 ± 0.32	91.02 ± 0.44	91.79 ± 0.41	91.17 ± 0.47	decay
$2 * 8$	90.94 ± 0.56	90.78 ± 0.39	88.76 ± 0.65	83.50 ± 1.80	84.10 ± 3.28	86.79 ± 1.23	none
$2 * 4$	56.48 ± 31.54	70.89 ± 17.56	72.05 ± 8.34	52.72 ± 8.35	49.19 ± 12.36	66.78 ± 7.11	dropout
$4 * 632$	97.94 ± 0.10	97.96 ± 0.05	98.15 ± 0.04	97.92 ± 0.07	97.90 ± 0.05	97.87 ± 0.12	dropout
$4 * 425$	97.81 ± 0.05	97.87 ± 0.07	98.03 ± 0.05	97.86 ± 0.07	97.85 ± 0.05	97.79 ± 0.08	dropout
$4 * 280$	97.70 ± 0.10	97.68 ± 0.07	98.01 ± 0.08	97.63 ± 0.10	97.69 ± 0.08	97.43 ± 0.07	dropout
$4 * 185$	97.50 ± 0.09	97.44 ± 0.12	97.88 ± 0.11	97.47 ± 0.12	97.39 ± 0.12	96.97 ± 0.22	dropout
$4 * 119$	97.23 ± 0.09	97.16 ± 0.16	97.71 ± 0.17	97.02 ± 0.10	97.12 ± 0.15	96.48 ± 0.34	dropout
$4 * 76$	96.70 ± 0.15	96.74 ± 0.16	97.39 ± 0.09	96.40 ± 0.18	96.50 ± 0.30	95.50 ± 0.16	dropout
$4 * 47$	95.98 ± 0.35	96.24 ± 0.13	96.50 ± 0.23	95.55 ± 0.28	95.60 ± 0.21	94.48 ± 0.33	dropout
$4 * 29$	95.04 ± 0.43	95.27 ± 0.19	95.01 ± 0.27	93.78 ± 0.48	93.92 ± 0.48	92.77 ± 0.52	decay
$4 * 17$	93.34 ± 0.53	93.32 ± 0.61	92.59 ± 0.25	90.61 ± 0.44	90.65 ± 0.53	90.03 ± 0.61	none
$115 * 115$	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	**Tie:** none/decay/dropout/full/hidden/output
$91 * 91$	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	**Tie:** none/decay/dropout/full/hidden/output
$71 * 71$	11.01 ± 0.03	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	11.02 ± 0.00	**Tie:** none/decay/dropout/full/hidden/output
$56 * 56$	11.01 ± 0.03	11.02 ± 0.00	10.93 ± 0.28	11.02 ± 0.00	10.90 ± 0.36	11.02 ± 0.00	**Tie:** none/decay/full/output
$44 * 44$	10.92 ± 0.27	11.02 ± 0.00	10.90 ± 0.35	10.75 ± 0.42	10.92 ± 0.27	11.02 ± 0.00	**Tie:** decay/output
$35 * 35$	10.84 ± 0.36	10.65 ± 0.54	10.54 ± 0.60	10.83 ± 0.37	10.81 ± 0.43	10.71 ± 0.47	**Tie:** none/full
$32 * 32$	10.57 ± 0.56	10.90 ± 0.36	10.62 ± 0.48	10.56 ± 0.57	10.63 ± 0.59	11.00 ± 0.04	output
$31 * 31$	10.78 ± 0.44	10.51 ± 0.50	10.55 ± 0.59	10.35 ± 0.56	10.66 ± 0.52	11.02 ± 0.00	output
$30 * 30$	10.73 ± 0.40	10.61 ± 0.50	10.67 ± 0.52	10.79 ± 0.45	11.01 ± 0.03	10.90 ± 0.36	hidden
$29 * 29$	10.72 ± 0.40	10.59 ± 0.52	10.50 ± 0.74	10.91 ± 0.27	10.64 ± 0.53	10.67 ± 0.52	full
$28 * 28$	10.67 ± 0.54	10.39 ± 0.53	10.64 ± 0.58	10.51 ± 0.60	10.66 ± 0.54	10.92 ± 0.27	output
$27 * 27$	10.56 ± 0.54	10.25 ± 0.50	10.46 ± 0.57	10.72 ± 0.46	10.42 ± 0.61	10.93 ± 0.28	output
$26 * 26$	16.78 ± 18.86	10.13 ± 0.71	10.19 ± 0.45	10.70 ± 0.48	10.56 ± 0.55	16.18 ± 15.52	none
$25 * 25$	10.37 ± 0.52	10.84 ± 0.37	10.49 ± 0.53	10.56 ± 0.46	10.84 ± 0.37	10.84 ± 0.36	**Tie:** decay/hidden/output
$24 * 24$	10.63 ± 0.43	10.27 ± 0.51	10.92 ± 0.27	10.70 ± 0.48	10.35 ± 0.75	10.93 ± 0.28	**Tie:** dropout/output
$23 * 23$	10.50 ± 0.53	10.61 ± 0.47	10.71 ± 0.49	10.44 ± 0.55	10.69 ± 0.49	10.89 ± 0.39	output
$22 * 22$	10.00 ± 0.82	10.33 ± 0.75	10.10 ± 0.49	10.58 ± 0.73	10.40 ± 0.46	10.98 ± 0.05	output
$21 * 21$	10.53 ± 0.59	10.17 ± 0.69	10.39 ± 0.48	10.61 ± 0.51	10.42 ± 0.57	10.80 ± 0.45	output
$20 * 20$	10.13 ± 0.45	10.69 ± 0.48	10.19 ± 0.54	10.59 ± 0.74	10.52 ± 0.51	11.02 ± 0.00	output
$19 * 19$	11.34 ± 3.02	17.24 ± 20.88	11.50 ± 2.98	10.46 ± 0.55	10.46 ± 0.58	10.61 ± 0.48	decay
$18 * 18$	10.48 ± 0.51	10.22 ± 0.74	10.18 ± 0.39	11.20 ± 2.74	10.11 ± 0.47	18.87 ± 17.87	output
$17 * 17$	16.18 ± 19.01	10.17 ± 0.69	10.37 ± 0.50	12.76 ± 4.35	24.56 ± 26.11	12.40 ± 3.88	hidden
$16 * 16$	11.14 ± 3.18	23.76 ± 25.24	10.13 ± 0.71	23.52 ± 23.73	18.04 ± 18.37	23.64 ± 20.87	decay
$15 * 15$	11.38 ± 3.05	11.57 ± 2.63	20.06 ± 18.82	26.67 ± 22.60	12.55 ± 4.36	36.52 ± 26.21	output
$14 * 14$	23.18 ± 26.84	12.83 ± 5.34	12.66 ± 8.67	13.26 ± 4.69	25.31 ± 26.00	32.08 ± 22.41	output
$13 * 13$	17.16 ± 21.46	11.91 ± 3.37	12.49 ± 4.32	18.68 ± 18.01	16.21 ± 14.60	26.49 ± 21.26	output
$12 * 12$	10.23 ± 0.62	13.14 ± 4.54	11.25 ± 3.64	22.61 ± 18.67	10.84 ± 3.03	27.06 ± 23.59	output
$11 * 11$	10.81 ± 2.37	11.14 ± 2.46	11.12 ± 3.18	36.10 ± 26.46	20.67 ± 18.70	44.24 ± 22.97	output
$10 * 10$	17.23 ± 21.59	12.26 ± 4.84	16.03 ± 7.53	17.68 ± 11.81	19.85 ± 15.51	26.20 ± 18.87	output
$9 * 9$	20.42 ± 18.24	13.06 ± 5.05	14.05 ± 6.51	18.39 ± 11.12	18.32 ± 13.57	24.06 ± 17.72	output
$8 * 8$	15.00 ± 6.66	19.51 ± 17.65	11.69 ± 3.93	25.16 ± 11.06	14.70 ± 6.38	27.96 ± 17.22	output
$7 * 7$	20.99 ± 20.96	15.44 ± 6.46	15.48 ± 7.21	25.02 ± 14.61	20.14 ± 12.01	44.32 ± 17.95	output
$6 * 6$	20.79 ± 19.33	20.30 ± 17.55	20.44 ± 12.59	29.55 ± 7.78	24.48 ± 12.64	23.20 ± 17.23	full
$5 * 5$	14.93 ± 5.22	17.30 ± 12.10	18.27 ± 13.38	25.70 ± 3.52	19.07 ± 9.26	25.08 ± 14.18	full
$4 * 4$	30.30 ± 21.80	34.51 ± 28.07	17.98 ± 8.90	26.77 ± 10.55	17.79 ± 7.12	33.09 ± 15.13	decay
$3 * 3$	26.04 ± 17.78	28.45 ± 19.36	34.73 ± 14.75	22.82 ± 8.99	15.81 ± 4.84	28.35 ± 15.06	dropout
$2 * 2$	23.54 ± 17.00	27.24 ± 14.06	24.95 ± 15.18	19.81 ± 5.41	18.93 ± 4.39	29.36 ± 10.44	output
$1 * 1$	18.84 ± 3.85	18.76 ± 3.94	18.52 ± 5.39	23.20 ± 2.04	13.07 ± 3.63	22.40 ± 1.63	full
$632 * 4$	9.90 ± 0.82	9.69 ± 0.36	10.11 ± 0.43	10.13 ± 0.70	9.72 ± 0.65	10.51 ± 0.48	output
$425 * 4$	9.98 ± 0.53	10.21 ± 0.91	10.00 ± 0.38	10.11 ± 0.61	10.01 ± 0.63	10.79 ± 0.35	output
$280 * 4$	10.32 ± 0.50	9.97 ± 0.81	9.89 ± 0.15	10.08 ± 0.60	10.41 ± 0.78	10.51 ± 0.47	output
$185 * 4$	10.23 ± 0.71	10.35 ± 0.51	9.87 ± 0.53	10.20 ± 0.75	10.47 ± 0.54	10.65 ± 0.43	output
$119 * 4$	9.88 ± 0.15	10.17 ± 0.69	9.87 ± 0.54	10.10 ± 0.61	10.00 ± 0.81	10.29 ± 0.43	output
$76 * 4$	10.08 ± 0.67	10.40 ± 0.48	9.95 ± 0.55	10.00 ± 0.55	9.83 ± 0.14	10.61 ± 0.47	output
$47 * 4$	10.03 ± 0.52	10.03 ± 0.70	9.80 ± 0.65	9.97 ± 0.63	10.02 ± 0.32	10.59 ± 0.47	output
$29 * 4$	9.96 ± 0.75	10.05 ± 0.35	10.21 ± 0.65	10.18 ± 0.52	10.04 ± 0.87	10.83 ± 0.36	output
$17 * 4$	10.39 ± 0.51	10.11 ± 0.68	9.95 ± 0.65	10.34 ± 0.54	10.09 ± 0.70	10.57 ± 0.55	output
$939 * 2$	10.12 ± 0.44	9.82 ± 0.52	9.81 ± 0.52	10.20 ± 0.51	10.47 ± 0.58	10.64 ± 0.44	output
$588 * 2$	9.83 ± 0.42	10.30 ± 0.45	10.24 ± 0.51	10.24 ± 0.49	9.97 ± 0.81	10.18 ± 0.40	decay
$354 * 2$	10.05 ± 0.33	10.08 ± 0.69	10.08 ± 0.71	10.11 ± 0.46	9.99 ± 0.64	10.61 ± 0.47	output
$214 * 2$	9.99 ± 0.65	9.89 ± 0.75	10.00 ± 0.61	10.17 ± 0.43	10.55 ± 0.53	10.28 ± 0.48	hidden
$115 * 2$	9.81 ± 0.58	10.12 ± 0.42	10.27 ± 0.49	9.99 ± 0.61	10.19 ± 0.65	10.54 ± 0.44	output
$59 * 2$	9.91 ± 0.53	9.75 ± 0.57	10.07 ± 0.36	10.26 ± 0.59	10.25 ± 0.50	10.31 ± 0.44	output
$30 * 2$	10.08 ± 0.32	9.98 ± 0.14	9.90 ± 0.54	9.99 ± 0.61	10.28 ± 0.46	10.58 ± 0.50	output
$16 * 2$	9.89 ± 0.53	9.89 ± 0.55	9.88 ± 0.16	10.30 ± 0.77	10.11 ± 0.46	10.43 ± 0.47	output
$8 * 2$	9.50 ± 0.57	10.09 ± 0.68	10.21 ± 0.75	10.72 ± 0.40	10.16 ± 0.72	10.20 ± 0.40	full
$4 * 2$	11.10 ± 2.68	11.86 ± 3.58	13.58 ± 6.96	13.38 ± 5.09	13.78 ± 5.51	14.35 ± 5.18	output
$2048 * 1$	10.09 ± 0.67	10.01 ± 0.37	9.78 ± 0.84	10.12 ± 0.47	9.91 ± 0.56	10.47 ± 0.45	output
$1024 * 1$	9.98 ± 0.35	10.29 ± 0.58	9.88 ± 0.53	10.17 ± 0.86	10.02 ± 0.61	10.51 ± 0.47	output
$512 * 1$	10.00 ± 0.83	10.17 ± 0.51	9.98 ± 0.35	10.22 ± 0.51	9.87 ± 0.55	10.61 ± 0.40	output
$256 * 1$	9.91 ± 0.53	9.88 ± 0.73	10.22 ± 0.50	9.73 ± 0.64	10.10 ± 0.34	10.61 ± 0.42	output
$128 * 1$	10.09 ± 0.44	10.10 ± 0.71	9.83 ± 0.42	9.91 ± 0.75	10.24 ± 0.46	10.32 ± 0.43	output
$64 * 1$	9.89 ± 0.75	10.23 ± 0.53	10.07 ± 0.69	10.25 ± 0.73	10.26 ± 0.48	10.62 ± 0.43	output
$32 * 1$	10.15 ± 0.42	9.65 ± 0.79	10.22 ± 0.74	10.15 ± 0.67	9.80 ± 0.54	10.47 ± 0.43	output
$16 * 1$	9.98 ± 0.62	9.70 ± 0.76	10.23 ± 0.67	10.24 ± 0.48	9.90 ± 0.53	10.46 ± 0.42	output
$8 * 1$	10.00 ± 0.53	10.01 ± 0.36	9.94 ± 0.84	10.44 ± 0.55	10.34 ± 0.51	10.43 ± 0.47	**Tie:** full/output
$4 * 1$	10.80 ± 2.18	10.05 ± 0.62	9.91 ± 0.53	10.48 ± 0.52	10.10 ± 0.46	10.55 ± 0.45	none
$2 * 1$	14.69 ± 4.95	16.78 ± 5.24	14.73 ± 7.21	16.75 ± 5.34	9.58 ± 0.70	16.89 ± 5.48	output

TABLE VI
TRAINING HYPERPARAMETERS

Parameter	Value
Optimizer	AdamW
Learning Rate	0.0001
Betas	(0.9, 0.999)
Epsilon	1e-8
Weight Decay	0 (unused) — Weight Decay 0.0001 (`decay` mode)
Dropout	0 (unused) — Dropout 0.1 (`dropout` mode)
Batch Size	256
Meta-Loops (Epochs)	100
Dataset	MNIST

CODE, DATASET, AND REPRODUCTION

Code Repository

- **GitHub**:
  https://github.com/tcotten-scrypted/persistent-stochastic-ablation-mlp
- **License**: Apache License 2.0
- **Dependency Management**: Poetry (https://python-poetry.org/)
- **Framework**: PyTorch
- **Supported Environments**:
  - CPU (fallback)
  - CUDA (GPU acceleration)
  - Metal (Apple M1/M2/M3)
- **Core Training Commands**:

```
poetry run train
poetry run train -- --arch "[1*512]" --ablation-mode full
```

- **Visualization Commands**:

```
poetry run make-design-space-figure
poetry run make-figure-heatmaps
poetry run make-convergence-plots --targets "..."
poetry run make-architecture-table
poetry run make-trial-accuracy-table
poetry run regime-classifier
```

- **AWS SageMaker Utilities**:

```
poetry run sagemaker-estimate-storage
poetry run sagemaker-get-training-logs
poetry run sagemaker-logs-parser
poetry run sagemaker-estimate-costs
```

Dataset

- **Dataset Used**:
  MNIST (https://ossci-datasets.s3.amazonaws.com/mnist/)
- **Download Method**: Automatically downloaded to `dataset/` on first run
- **License**: Public/open dataset
- **Validation ZeroR Baseline**: 11.02% (dynamically computed)
- **Data Split**: 50,000 training, 10,000 validation, 10,000 test
- **Split Method**: Original 60,000 training set randomly split into 50,000 training + 10,000 validation (static seed: 1337), 10,000 test set held independent
- **Preprocessing**: Normalized pixel values to [0,1] range

AWS SageMaker Infrastructure

- **Training Jobs**: 588 total jobs across 98 architectures times 6 ablation modes
- **Instance Type**: ml.g4dn.xlarge (4 vCPUs, 16GB RAM, 1 GPU)
- **Storage Requirements**:
  - Model artifacts: 2.5GB total
  - CloudWatch logs: 15GB total
  - S3 bucket organization: `psa-simplemlp-results/`
- **CloudWatch Logging**:
  - Log group: `/aws/sagemaker/TrainingJobs`
  - Log streams: One per training job with format `psa-<arch>-<mode>-<timestamp>`
  - Log retention: 14 days (configurable)
  - Log parsing: Extracts meta-loop progress, LKG tracking, validation accuracy
- **Job Naming Convention**: `psa-<depth>x<width>-<mode>-<timestamp>`
- **Actual Cost**: $640.32 total (869.5 billable hours at $0.7364/hr)
- **Concurrency Efficiency**: 27.9 hours wall-clock time (30x speedup)
- **Longest-Running Job**: 2048*1 architecture with dropout mode (27.9 hours)

Reproducibility

- **Reproduction Guide**: `REPRODUCTION.md`
- **Configuration List**: `reproduction/configurations.txt` (98 architectures)
- **Trial Structure**: 6 ablation modes per architecture, 10 trials per mode
- **Random Seed Handling**: Hardware RNG per trial
- **Environment Setup**: `poetry install` for all dependencies

Artifacts

- **Model Checkpoints**: Saved in `models/`
- **Results**:
  - `results/psa_simplemlp_summary.md`
  - `results/psa_simplemlp_trials.md`
  - `results/psa_simplemlp_trials_lkg_growth.md`
  - `results/psa_simplemlp_trials_convergence.md`
  - `results/sagemaker_cost_report.json`
- **Figures and Tables Generation**:
  - `results/SimpleMLP_Heatmap_Ablation_Effects.png`
  - `results/SimpleMLP_Heatmap_Baseline_Performance.png`
  - `results/SimpleMLP_Heatmap_Instability.png`
  - `results/SimpleMLP_Heatmap_Parameter_Matching.png`
  - `results/SimpleMLP_Heatmap_Regimes.png`
  - `results/SimpleMLP_Heatmap_Winning_Strategy.png`
  - `results/SimpleMLP_Plot_Convergence.png`
  - `results/SimpleMLP_Testing_Design_Space.png`
- **CloudWatch Logs**: Downloaded to `results/logs/<job_name>/`

Training and Evaluation Protocol

- **Architecture Specification**: Defined via `--arch` (e.g., `"[2*939]"`)
- **Training Length**: 100 meta-loops (1 epoch per loop)
- **Evaluation Metric**: Peak validation classification accuracy on MNIST
- **LKG Tracking**: Last Known Good model state based on validation accuracy
- **Ablation Modes Supported**:
  - `none` (Control)
  - `decay` (Weight Decay)
  - `dropout` (Dropout)
  - `full` (Full Ablation)
  - `hidden` (Hidden Ablation)
  - `output` (Output Ablation)
- **Training Regimes**: Automatically classified into four categories:
  - I. Beneficial Regularization (Over-parameterized)
  - II. Optimally Sized (At-Capacity)
  - III. Chaotic Optimization (Unstable)
  - IV. Architectural Failure (Untrainable)